
CAMPUS
DE EXCELENCIA
INTERNACIONAL

**Graduado en Ingeniería Informática**

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

# TRABAJO DE FIN DE GRADO

## Sistema de Generación de la Web de beBee para usuarios anónimos

Autor: Ovidiu Dan Pop

Director: Santiago Rodríguez de la Fuente

MADRID, JULIO 2016

# ÍNDICE

## ÍNDICE DE ILUSTRACIONES

**RESUMEN**

El proyecto consiste en la creación de un sistema distribuido y escalable para la generación de ficheros HTML estáticos que sustituyen la parte pública de la aplicación web de la red social beBee para aumentar la velocidad de carga de dichas páginas y con ello mejorar el posicionamiento de la web en buscadores como Google.

El desarrollo del sistema se ha realizado utilizando el lenguaje de programación Python 2.7 debido a la alta velocidad que ofrece y la elevada cantidad de librerías disponibles para él.

Tras barajar distintas opciones se concluyó que ésta solución sería la más rápida en cuanto a la velocidad de carga de la web, la que menos recursos humanos necesitaría y la que se llevaría a cabo en menos tiempo. Se decidió apostar por ella pese a que requeriría más capacidad de procesamiento (CPU) y más espacio de almacenamiento, debido a que el coste del almacenamiento no es elevado.

**ABSTRACT**

The project involves the development of a distributed and scalable system for generating HTML static files replacing the public part of the web application of social network beBee to increase the loading speed of these pages and thereby improve the web positioning on search engines like Google.

The development of the system was performed using the programming language Python 2.7 due to the high speed offered and the high number of libraries available for it.

After considering various options it was concluded that this would be the quickest solution regarding the loading speed of the web, the less human resources needed and which would be done in less time. It was decided to bet on it even though it would require more processing power (CPU) and more storage space, because the cost of storage is not high.

## 1. INTRODUCCIÓN

Este proyecto se ha elaborado para la empresa beBee Affinity Social Network, SL siendo una empresa que ofrece como servicio una red social parecida a LinkedIn que pretende unificar el perfil profesional y personal de manera que no se tenga que separar la experiencia laboral de los gustos y las aficiones de los usuarios, buscando maximizar la afinidad para el usuario, siendo el contenido que se muestra en el muro de cada uno de interés y relevante para el mismo.

Dentro de la empresa pertenezco al departamento de desarrollo ocupando el cargo de Software Developer.

Mediante este proyecto la empresa perseguía dos objetivos principales. Por un lado se buscaba separar la aplicación web pública de la aplicación privada dirigida a los usuarios registrados. Por otro lado se pretendía aumentar el tráfico cualificado [1] (entendiéndose por tráfico cualificado posibles usuarios de dicha web) desde buscadores y aumentar el número de registros de tráfico orgánico (el tráfico proveniente de las páginas de resultados de un buscador) que visitan beBee por primera vez, mediante la creación de una nueva aplicación web a modo de directorio que favorezca el internal linking y de este modo el rastreo e indexación de la misma, generando nuevas plantillas orientadas a SEO.

El internal linking [2] puede definirse como cualquier enlace dentro de un dominio (en este caso el de beBee) que redirija a otra página dentro del mismo dominio.

El SEO [3] es un conjunto de técnicas destinadas a mejorar el posicionamiento en buscadores mediante la optimización de la web para su aparición en las primeras posiciones de los motores de búsqueda para mejorar la visibilidad de una página web. Mediante este proyecto y las plantillas orientadas a este fin se pretende aumentar la relevancia y la visibilidad del contenido almacenado en beBee en buscadores como Google.

De este modo se separa la web interna, a la que solo pueden acceder los usuarios registrados y por tanto inaccesible para los motores de búsqueda, de la web pública, de total acceso.

El proyecto está orientado al desarrollo de un sistema para la generación de ficheros html estáticos a partir de la información almacenada en la base de datos de la empresa y debe ser capaz de exportar cualquier información a voluntad de la dirección.

El sistema se ha desarrollado en Python [4] y dispone de una amplia configuración, tanto dentro de un fichero de configuración (en el que se puede definir el número de máquinas entre las que se repartirá la carga y el número de maquina actual, entre otras opciones), como mediante parámetros, a través de la cual se debe indicar al sistema qué se desea hacer y cómo debe hacerlo.

El sistema está capacitado para la generación de las fichas de los usuarios registrados en la aplicación, los distintos índices disponibles para navegar hacia las fichas de usuarios,

el mapa del sitio web (sitemap) correspondiente a la estructura de enlaces generada y las fichas de las habilidades validadas por la empresa. Estas mismas características se aplican también para las ofertas de empleo de la red social, para los distintos grupos y para las publicaciones realizadas por los usuarios. Además se han implementado en scripts auxiliares varias funcionalidades auxiliares, siendo éstas, entre otras, la generación de la página principal y la generación de una lista de usuarios ordenados por la fecha de creación de los perfiles.

La ejecución del sistema se puede realizar de dos modos diferentes, el primero de ellos generará todo el conjunto de fichas con las opciones indicadas a partir de la información de la base de datos mientras que el segundo comprobará la integridad de las fichas generadas verificando si ha habido cambios en la información de la base de datos desde la fecha de generación de la ficha o desde la última comprobación, regenerando la ficha afectada si es necesario. Se entrará en más detalle en capítulos posteriores.

## 2. CONTEXTO Y REQUISITOS DEL PROYECTO

Antes de comenzar resulta conveniente definir el contexto en el cual se realiza el proyecto. Como se ha mencionado anteriormente, dicho proyecto se realiza para la empresa beBee Affinity Social Network, cuyas características principales se pueden ver a continuación:


*Ilustración 1 Infografía bebee*

Como se puede observar, la red social tiene información relevante a usuarios, ofertas de empleo, grupos y publicaciones. Estos elementos son fundamentales para la realización del proyecto ya que lo que se busca es que dicha información aparezca en las primeras posiciones de los resultados de buscadores como Google (mejorar el SEO de la web).

Además cabe destacar que la red social cuenta con aproximadamente doce millones de usuarios una cantidad que hay que tener en cuenta para el tiempo de generación.

Inicialmente se tiene una única aplicación web en PHP gestiona tanto la parte pública como la parte privada de la web, la cual es demasiado lenta para los robots de indexación de Google y conlleva a una baja indexación de la información en el buscador.

Para hacer posible el objetivo que se persigue se decide separar la parte pública de la web mediante la realización de este proyecto.

Dado que la velocidad de carga de la página [5] es un factor crítico para el SEO se busca un sistema lo más rápido posible. Debido a que la información que se encuentra en la red social no cambia mucho (los perfiles de usuarios no se actualizan muy a menudo, las ofertas de empleo apenas cambian una vez creadas) permite realizar la web pública como ficheros html estáticos, añadiendo un sistema que regenere aquellos ficheros en los que se producen cambios o los que se han creado nuevos.

La estructura de la web por la que se ha optado es una estructura de fichas y directorios. Las fichas corresponden con la información relevante a los cuatro elementos fundamentales presentes en la red social (usuarios, ofertas de empleo, grupos, publicaciones) y los directorios son los distintos índices por los que se puede acceder a una ficha en concreto. Estos índices aparte de permitir la navegación están orientados a los buscadores, ya que para el uso humano también se dispone de un buscador.

Se tienen dos tipos de índices, el primero es de tipo árbol, para la navegación por rangos al hacer búsquedas por nombre, como puede pasar con los nombres de usuarios o los nombres de grupos, el segundo es de tipo paginado, para la navegación por la lista de ofertas de empleo o por la lista de publicaciones.

Con los índices se pretende una mejor indexación de las fichas clasificándolas en ciertas etiquetas. Por ejemplo si se tienen el índice de ofertas de empleo de la categoría informática y la ubicación Madrid se pretende que al realizar la búsqueda “ofertas de empleo de informático en Madrid” se a más probable que aparezcan las fichas de ese índice entre los primeros resultados del buscador.

Al generar las fichas se extrae la información necesaria de la base de datos principal de la empresa, mientras que para la generación de los directorios se necesita información de la generación de las fichas, por lo que se dispone de una base de datos centralizada en MySQL para el proyecto. Esta información también es necesaria para la generación de los mapas de navegación del sitio web (sitemaps).

En cuanto a los requisitos del proyecto, éste debe ser capaz de generar las fichas, directorios y sitemaps de todos los elementos descritos anteriormente, debe proporcionar el mecanismo para la regeneración de los elementos que han sido modificados y para generar nos elementos nuevos sin tener que regenerar todas las fichas, la página web debe ser lo más rápida posible (por ello se generan los ficheros html estáticos, ya que solo se deben localizar y servir sin más operaciones), debe ser capaz de generar y trabajar con una gran cantidad de ficheros (del orden de millones) a una alta velocidad, y debe ser un sistema escalable ya que la red social está en crecimiento, con lo cual se tiene previsto aumentar la cantidad tanto de usuarios como de ofertas de empleo, grupos y publicaciones, por lo que se ha planteado un sistema distribuido.

Adicionalmente el sistema debe tener otras funcionalidades que no implica la generación de fichas y directorios, sino que son ficheros que son necesarios para completar la parte pública de la web, estas funcionalidades son:

- Generación de la página principal (Home) la cual contiene información referente a una muestra de usuarios, una muestra de los grupos y publicaciones. Ésta página se regenera cada cinco minutos.

- Un listado de usuarios ordenados por fecha de registro de manera que los últimos usuarios registrados aparezcan en la primera página.

- Un listado con los subdominios disponibles.

- Un listado con los países en los que se tiene presencia

- Internalización de las plantillas, función para obtener los textos a traducir y ser capaz de traducir las plantillas.

## 3. ESTADO DEL ARTE

Como trabajos previos de la memoria se investigó cómo había desarrollado estas características la competencia, siendo en este caso LinkedIn y Facebook, y se partió de su ejemplo.

Dado que se está buscando mejorar el SEO y la velocidad de carga de la web es uno de los factores que influyen en el posicionamiento, ya sea debido a que si la velocidad de carga es lenta los robots no tienen tiempo suficiente a rastrear toda la web para su posterior indexación en los buscadores o debido a que si la web no carga rápido la tasa de rebote es alta, se ha decidido que la web se serviría como ficheros HTML estáticos.

La tasa de rebote [6] se calcula dividiendo el número total de visitas que visualizan una sola página web entre número total de visitas a la página web. Un rebote se produce cuando un visitante abandona el sitio web después de haber visto una sola página web, en pocos segundos, es decir, sin navegar por la web. Por tanto, se desea tener una tasa de rebote lo más baja posible.

Otras optativas estudiadas fueron la aceleración de la aplicación web ya existente y la sustitución del sistema gestor de bases de datos, pero dichas alternativas serían más lentas de llevar a cabo y necesitarían de más personal, por lo que se ha decidido apostar por la solución propuesta en este proyecto invirtiendo más en capacidad de cómputo y almacenamiento debido a que se implementaría más rápidamente y utilizando menos recursos humanos.

El primer paso llevado a cabo fue elegir el lenguaje de programación a utilizar, y debido a su velocidad al tratar con ficheros y la posibilidad de multithreading [7] se ha optado por Python. Una vez decidido el lenguaje se investigó sobre qué versión se utilizaría, estando presentes las versiones 2.7 y 3.X. La versión 3 de Python es relativamente nueva e implementa nuevas funcionalidades, aunque la versión 2.7 está mucho más extendida y como consecuencia cuenta con una mayor cantidad de librerías y código reutilizable, por lo que se decidió hacer uso de dicha versión.

Dada la naturaleza del proyecto, fue esencial contar con un motor de plantillas rápido, compatible con Python y con un sistema de internacionalización de plantillas. Para estas tareas se ha utilizado CheetahTemplate [8], un motor de plantillas escrito en Python que cuenta con una alta velocidad y la flexibilidad de Python, ya que compila las plantillas dicho lenguaje. También proporciona la posibilidad de acceder a funciones y objetos de Python haciendo más sencilla la tarea.

Otras tecnologías utilizadas han sido MySQL [9] para la base de datos de apoyo de este proyecto, junto con SQLite utilizado para la creación de bases de datos temporales.

Aunque la empresa puede considerarse como una Startup ya que tiene poco más de un año de vida, ya cuenta con más de 10 millones de usuarios, y tiene publicadas más de un millón de ofertas de empleo. Si a esto se le suman los grupos con los que cuenta y la

cantidad de publicaciones dentro de la red social, además de los índices que se van a generar, la cantidad de ficheros manejados es muy elevada. Debido a esto se ha llevado a cabo una investigación previa respecto a la estructura del proyecto y el algoritmo de sharding [10], concepto que se explicará posteriormente, que se implementó para repartir dicha cantidad de ficheros entre varias máquinas. En un principio se pensó en 100 shards, aunque durante el desarrollo se llegó a la concusión de aumentarlo a 256. El algoritmo se explicará más adelante, mientras que la estructura de las máquinas que almacenan los archivos es la siguiente:

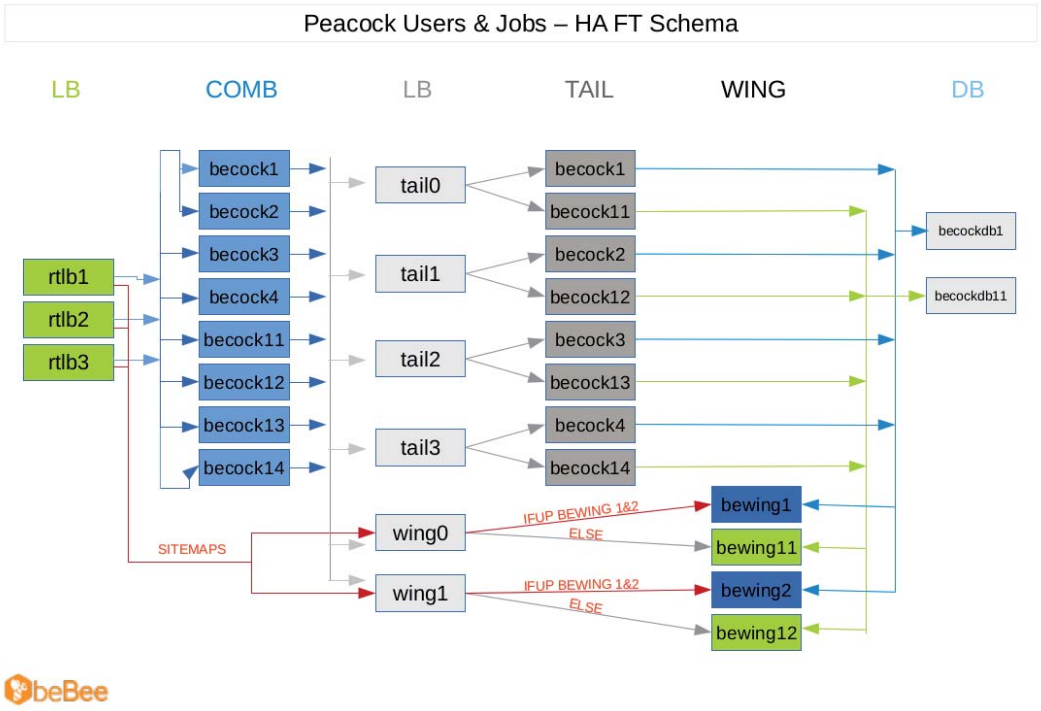


*Ilustración 2 Estructura de máquinas*

En la ilustración se observa que se dispone de balanceadores de carga (“LB”) [11], estos funcionan en la capa 7, a nivel de aplicación. También se dispone de dos balanceadores de carga que no aparecen en la ilustración entre la conexión a internet y la primera capa de los balanceadores de carga que aparecen en la imagen, aunque estos balanceadores funcionan en la capa 4, a nivel de la capa de transporte TCP, y que tienen el protocolo SSL activo [12].

Como se puede observar se cuenta con un circuito alternativo, es decir, se tienen máquinas replicadas tanto para servir los ficheros HTML como para la base de datos para evitar que el sistema falle si se cae una máquina.

Al tratarse de un proyecto con un objetivo específico, llevado a cabo para una empresa específica, se han seguido los requisitos proporcionados por dicha empresa, aunque

durante la fase de desarrollo se ha investigado sobre mejoras de rendimiento para alcanzar las especificaciones.

# 4. DESARROLLO

## 4.1 Estructura del Proyecto.

La estructura del proyecto es la siguiente:

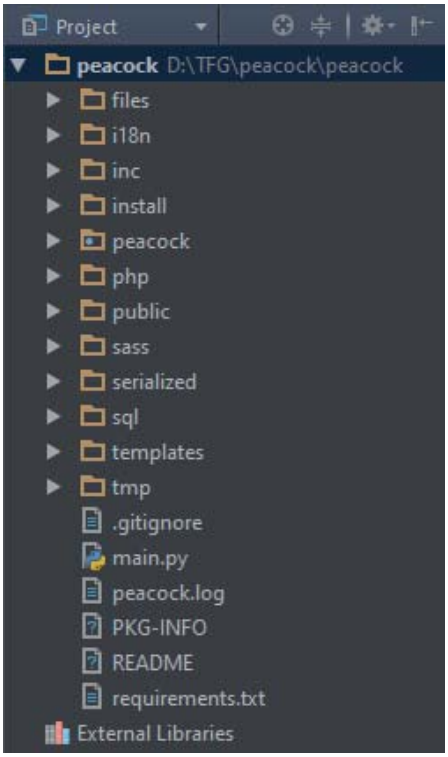


*Ilustración 3 Estructura general del proyecto*

- **files**: Contiene los ficheros generados por el programa, ordenados en subdirectorios de diferentes niveles según el shard al que pertenecen.

- **i18n**: Contiene los ficheros necesarios para la internalización de las plantillas, esto es, los ficheros de traducción de los diferentes idiomas.

- **inc**: Contiene los ficheros relacionados con la base de datos generados cuando la aplicación es lanzada en modo incremental.

- **install**: Contiene un script de post-instalación mediante el cual se copian los ficheros .php de los mensajes de error 404 (página no encontrada) y se lanza el script de traducción de plantillas del que se hablará más adelante.

- **peacock**: Contiene todas las clases y la lógica de negocio del proyecto. Se entrará en detalles más adelante.

- **php**: Contiene los ficheros .phtml de los mensajes de error 404.

- **public**: Contiene las imágenes, fuentes, hojas de estilo .css y ficheros javascript que se utilizan en el proyecto.

- **sass**: Contiene los ficheros necesarios para generar los ficheros de estilo .css

- **serialized**: Contiene bases de datos de muestra que se utilizan para mostrar las secciones de empleos similares, compañías del sector, etc. Esto se hace debido a que se quiere una aplicación rápida, con lo cual no se aplican los algoritmos de afinidad que se utilizan en la versión para usuarios registrados de la aplicación.

- **sql**: Contiene las funciones y procedimientos SQL que se utilizan en el proyecto, así como el esquema de la base de datos en la que se apoya el proyecto. De este último fichero se hablará más adelante.

- **templates**: Contiene las plantillas y partials utilizadas en el proyecto en el formato usado por Cheetah Templates .tmpl. Aquí se encuentran dos subdirectorios, en uno encontramos las plantillas originales en inglés y en otro encontramos las plantillas en cada idioma generadas por el script de traducción.

- **tmp**: Contiene ficheros temporales que se crean durante la ejecución del proyecto.

- **main.py**: Es el script principal que controla el proyecto. Se describirá en detalle más adelante.

- **peacock.log**: Es el fichero en el que se almacena el log de la ejecución de la aplicación.

- **requirements.txt**: Contiene las dependencias de Python necesarias para la ejecución del proyecto, así como sus versiones.

A continuación se tratará con más detalle el directorio “peacock” que contiene el núcleo del proyecto:

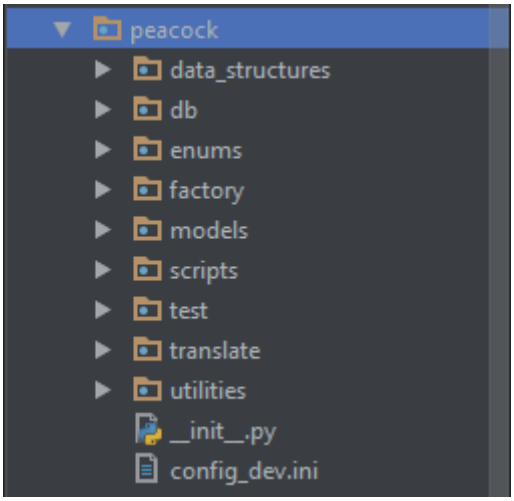


*Ilustración 4 Estructura del directorio peacock*

- **data-structures**: Aquí encontramos estructuras de datos, como listas de objetos y rangos que se utilizan para la creación de los distintos índices.

- **db**: Contiene las clases que interactúan con las bases de datos, tanto la base de datos principal de la empresa en “MainDB”, como la base de datos de apoyo del proyecto en MySQL.

- **enums**: Contiene los ficheros de tipo enumerado en los que se definen diferentes conjuntos de constantes, como los posibles tipos de árbol o los posibles tipos de ofertas de trabajo, entre otros.

- **factory**: Aquí se encuentran las funciones que generan los ficheros estáticos html pre-comprimidos. Se distinguen tres tipos de generadores: generador de directorio, en los que se obtienen las hojas del árbol (las fichas finales); generador de índices, en los que se obtienen los ficheros de navegación para alcanzar las fichas finales; y generador de sitemaps, en los que se obtienen los ficheros que describen el mapa de la página web de un directorio dado.

- **models**: Contiene todos los modelos de objetos utilizados en el proyecto.

- **scripts**: Contiene scripts auxiliares como podría ser el script de generación de las páginas de países, o el script de generación del índice de subdominios, entre otros.

- **test**: Contiene los tests unitarios.

- **translate**: Contiene las funciones relacionadas con la traducción de las plantillas.

- **utilities**: Contiene funciones auxiliares utilizadas en el proyecto y el controlador de la cola de interacción con la base de datos.

- **config-dev.ini**: Modelo de fichero de configuración del proyecto para desarrollo.

### 4.2 Configuración del Proyecto.

#### 4.2.1 Fichero de Configuración.

A continuación se mostrará y explicará el modelo del fichero de configuración del proyecto. Algunos datos se han tenido que censurar por motivos de confidencialidad.

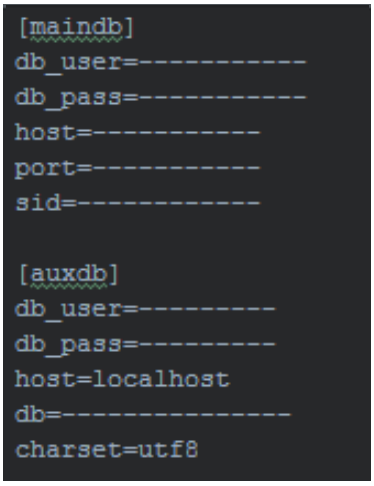


*Ilustración 5 Fichero de configuración 1*

Se define la conexión con las bases de datos, tanto la base de datos principal de la empresa "MainDB" como la base de datos de apoyo del proyecto en MySQL.

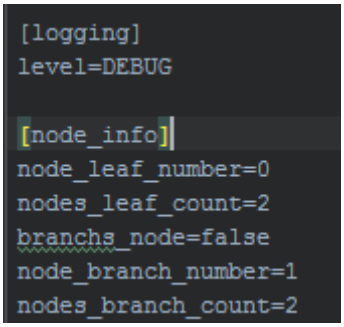


*Ilustración 6 Fichero de configuración 2*

Se define el tipo de logging que se desea.

En la información del nodo, el parámetro “node_leaf_number” indica el número de la máquina actual y el parámetro “nodes_leaf_count” indica el número total de máquinas en las que está distribuido el directorio que se va a ejecutar.

El parámetro “branchs_node” indica si se van a generar nodos hoja o nodos rama ya que la estructura de navegación será de tipo árbol. Con lo cual los nodos hoja son las fichas finales mientras que los nodos rama forman los índices.

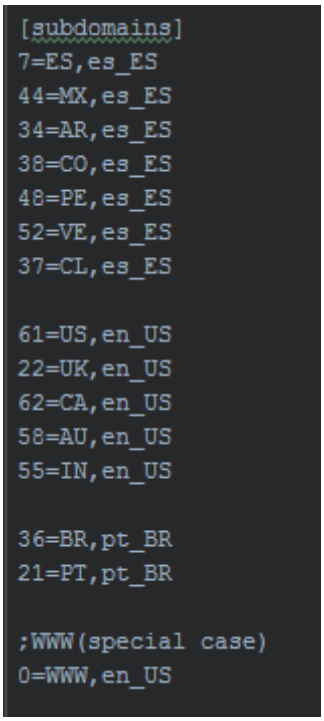


*Ilustración 7 Fichero de configuración 3*

Se configuran los subdominios disponibles en la aplicación, asignándole un identificador y un idioma a cada uno de ellos.

```
[std]
url=https://stdg.bebee.com

[inc]
path_files=/var/www/projects/peacock/inc
queue_id=0

[path_files]
path=/var/www/projects/peacock/files
```

*Ilustración 8 Fichero de configuración 4*

En la sección “std” se configura la dirección de la máquina CDN (Content Delivery Network) que servirá las imágenes en el proyecto. Este tipo de máquinas se utilizan para disminuir la velocidad de carga.

Dentro de “inc” tenemos la ruta en la que se guardarán los ficheros .sql.gz de los modos de ejecución  y se define el identificador de la cola a utilizar, ya que por tolerancia a fallos se dispone de 2 colas iguales. Se hablará más adelante de estas características.

El parámetro de “path_files” define el directorio donde se desean generar los ficheros html pre-comprimidos.

Asimismo en el mismo fichero se configuran también la ruta hacia los ficheros de traducción, la ruta hacia los ficheros de log, y la ruta de los partials de las plantillas.

```
[leaf_quality]
user=50
job=100
group=50
content=50

[video_domains]
youtube=www.youtube.com,youtu.be,m.youtube.com
vimeo=vimeo.com
```

*Ilustración 9 Fichero de configuración 5*

En la sección “leaf_cuality” se configura el valor mínimo que debe alcanzar una ficha para ser considerada de calidad. Por ejemplo, una ficha de usuario suma puntos si contiene una descripción, incluye foto o dispone de al menos 3 experiencias, entre otros factores.

En la sección de “video_domains” se configuran los dominios aceptados para la publicación de videos en la red social.

```
;max and min url limits for branchs
[branchs_limits]
per_branch_max=50
per_branch_min=30

;groups_broadcast (incremental mode)
[groups_broadcast_frequency]
frequency=1

;Home page params
[index]
num_buzzes=10
hives_position=4
users_position=8
num_hives_users=3
video_es=I9G3nkWwZS8
video_en=rxTq_OZA02Y
video_pt=FkNK0O9W5RM
next_pages_size=50
```

*Ilustración 10 Fichero de configuración 6*

En la sección de “branchs_limits” se configura el número máximo y mínimo de enlaces por rama dentro del árbol.

En la sección “groups_broadcast_frequency” se configura la frecuencia con la que se lanzará el incremental del directorio de grupos (ahora llamados colmenas en la red social) en días.

En la sección “index” se configuran los parámetros para la generación de la página principal. Dentro de dicha sección se tiene el número de buzzes que se muestran, la posición de las colmenas y usuarios, el número de usuarios por colmena a mostrar, las direcciones de los videos de presentación de la red social de youtube y el número de enlaces de las páginas siguientes al hacer click en “ver más actualizaciones”.

### 4.2.2 Configuración por Parámetros.

Los parámetros disponibles dependen del campo “branch_node” del fichero de configuración, ya que la generación de hojas no dispone de los mismos parámetros que la generación de ramas.

Parámetros de generación de ramas:

```
parser.add_argument("-t", "--tree",
                    choices=["LETTER", "HIVE_LETTER", "HIVE_PROVINCE", "HIVE_COUNTRY",
                             "HIVE", "HIVE_LOCATION", "LOCATION", "SKILLS_USER", "SKILLS_JOB",
                             "LANG_LETTER", "SITEMAP", "ALL"],
                    required=True,
                    help="Define the tree type.")

parser.add_argument("-s", "--subdomain",
                    choices=['ES', 'MX', 'AR', 'CO', 'PE', 'VE', 'CL', 'PT', 'BR',
                             'US', 'UK', 'CA', 'AU', 'IN', 'WWW', 'ALL'], required=True,
                    help="Define the subdomain.")

parser.add_argument("-m", "--sitemap",
                    choices=['LEAF', 'TREE'], help="""Define the type of sitemap if sitemap is selected in --tree
                                        option.""")

parser.add_argument("-p", "--sitemaptree",
                    choices=["LETTER", "HIVE_LETTER", "HIVE_PROVINCE", "HIVE_COUNTRY",
                             "HIVE", "HIVE_LOCATION", "LOCATION", "SKILLS_USER", "SKILLS_JOB",
                             "LANG_LETTER", "ALL"], help="""Define the tree type of sitemap if --sitemap option is
                                        selected.""")

parser.add_argument("-d", "--directory", choices=["USERS", "JOBS", "GROUPS", "CONTENTS", "GROUPS_BROADCAST"],
                    help="Define the import directory.", required=True)

parser.add_argument("-f", "--path", default=None,
                    help="Path for created files.")

parser.add_argument("-l", "--log", default=None,
                    help="Path for log file.")
```

*Ilustración 11 Configuración por parámetros 1*

- **Tree**: tipo de árbol a generar entre la lista disponible.

- **Subdomain**: subdominio para el que se quiere generar el árbol.

- **Sitemap**: selecciona el tipo de sitemap que se quiere realizar si en el parámetro tree se ha seleccionado sitemap.

- **Sitemaptree**: si el árbol seleccionado es sitemap y el tipo de sitemap es tree se selecciona el tipo de árbol para el que generar el sitemap.

- **Directory**: el directorio para el que se quiere generar el árbol. Se puede elegir entre usuarios, empleos, grupos, contenido (publicaciones en la red social) o difusiones dentro de los grupos.
- **Path**: ruta en la que se quiere generar los ficheros. Dicha ruta predomina sobre la ruta del fichero de configuración.

- **Log**: ruta en la que se quiere generar el fichero de log. Dicha ruta predomina sobre la ruta del fichero de configuración.

Parámetros de generación de hojas:

```
parser.add_argument("-i", "--init", type=int, choices=range(0, 256), metavar='Range in 0-255', required=True,
                    help="""Define the start code range. Allowed values are in range 0-255
                    corresponds to 0-ff in hexadecimal. (Value 0 means 00, 1 means 01 and so one)""")

parser.add_argument("-e", "--end", type=int, choices=range(0, 256), metavar='Range in 0-255', required=True,
                    help="""Define the end code range. Allowed values are in range 0-255
                    corresponds to 0-ff in hexadecimal. It has to be greater or equal than -i if defined.
                    (Value 0 means 00, 1 means 01 and so one)""")

parser.add_argument("-m", "--mode", choices=["FULL", "INC"], required=True,
                    help="Define the import mode.")

parser.add_argument("-d", "--directory", choices=["USERS", "JOBS", "GROUPS", "CONTENTS", "GROUPS_BROADCAST"],
                    required=True, help="Define the import directory.")

parser.add_argument("-f", "--path", default=None,
                    help="Path for created files.")

parser.add_argument("-l", "--log", default=None,
                    help="Path for log file.")
```

*Ilustración 12 Configuración por parámetros 2*

- **Init**: se tienen 256 shards, aquí se selecciona el shard en el que se desea empezar la generación.

- **End**: se selecciona el último shard a generar.

- **Mode**: modo de generación, el cual puede ser completo o incremental.

- **Directory**: el directorio para el que se quiere generar el árbol. Se puede elegir entre usuarios, empleos, grupos, contenido (publicaciones en la red social) o difusiones dentro de los grupos.

- **Path**: ruta en la que se quiere generar los ficheros. Dicha ruta predomina sobre la ruta del fichero de configuración.

- **Log**: ruta en la que se quiere generar el fichero de log. Dicha ruta predomina sobre la ruta del fichero de configuración.

### 4.3 Funcionalidades y Modos de Ejecución.

#### 4.3.1 Estructurado de Ficheros por Sharding Code

En el proyecto se necesita almacenar una gran cantidad de ficheros, con lo cual para su almacenamiento se planteó un sistema por sharding code, siendo esto una manera de clasificar los ficheros en fragmentos, a partir de ahora “shards”.

En un principio se pensó en 100 shards, aunque se vió que con ello había demasiados ficheros por shard y se decidió aumentarlo a 256, equivalente al máximo número con 2 dígitos en hexadecimal, desde 00 hasta FF.

En el caso de los usuarios se dispondrían de doce millones de fichas, que repartidas entre 100 shards en dos niveles acabarían siendo 1200 ficheros por directorio, y debido teóricamente las búsquedas se consideran pesadas al haber un número superior a 1000 ficheros por directorio, por temas de inodos e indirecciones [13].

Con esto se obtiene un proyecto que se puede distribuir entre diferentes máquinas para la generación de los ficheros en paralelo y sin solaparse, ya que se puede indicar de qué shard a qué shard se desea generar.

Además, al generar los índices se utiliza el shard y la configuración del número de máquinas en las que se distribuirá para saber si una página del índice se debe generar en la maquina actual o no, como se muestra a continuación:

```
# verify if branch will be saved in current node
if hex_to_int(sharding_code(subdomain.lower() + '_' + profile_list_uri)[
            :2]) % nodes_branch_count != node_branch_number:
    return
```

*Ilustración 13 Comprobación shard-máquina*

Se comprueba si el módulo entre el sharding code y el número de máquinas asignadas es distinto de la maquina actual, en cuyo caso no se genera el fichero.

Dentro del proyecto, para calcular el shard al que pertenece cada elemento, se utiliza el permalink de dicho elemento, siendo esto una URL permanente al artículo en cuestión.

Las funciones para calcular el sharding code son las siguientes:

```
def sharding_code(permalink):
    i = _pre_sharding(permalink)
    code = str(i)[-4:]  # this is like % 10 thousands (last 4 positions)

    return code


def _pre_sharding(permalink):
    m = hashlib.md5()
    m.update(permalink)
    md5 = m.hexdigest()

    return md5
```

*Ilustración 14 Funciones de obtención de sharding code*

A partir del permalink se obtiene el md5 en hexadecimal gracias a la librería hashlib de Python y se seleccionan los últimos 4 dígitos.

Se pensaron otras funciones de shard como podría ser murmur2 [14] pero se descartó debido a que no estaba implementada en el sistema de la "MainDB", mientras que la función md5 sí está implementada en ambos sistemas.

A partir de este código se ha decidido organizar el directorio de los ficheros en dos niveles, cuya ruta se obtiene con la siguiente función:

```
def path_from_code(sharding):
    initial = str(sharding)
    result = ''

    while initial != '':
        result += '/' + initial[:2]
        initial = initial[2:]

    return result
```

*Ilustración 15 Función "path_from_code"*

A partir del código se seleccionan los dos primeros dígitos y se concatenan a la barra, eliminando esos dígitos de la cadena. Dicho proceso se repite hasta que la cadena esté vacía y con ello se obtiene la ruta del shard donde se debe almacenar el archivo generado.

### 4.3.2 Reordenación de los Árboles Generados en los Índices

En un principio, para la generación de los índices, se utilizaba una estructura de árbol en la que se navegaba por rangos, de más generales a más específicos, con lo cual los primeros nodos del árbol eran los que estaban menos poblados.

Después del despliegue de la primera fase se observó que esta característica no era apropiada para los robots de indexación de Google ya que al detectar que una página tenía poco contenido la clasificaban como poco importante y no continuaban con la navegación hacia los siguientes nodos más poblados, y por tanto no llegaban a los nodos finales en los que se encontraba la información.

Para solventar este inconveniente se desarrolló un algoritmo que calcula el nivel del árbol que se está generando para poblar más los nodos superiores del árbol y dejar menos poblados los nodos inferiores.

El desarrollo del algoritmo es el siguiente:

```
limits = BranchsLimits()
per_branch_max = limits.per_branch_max
per_branch_min = limits.per_branch_min

t = count
n = ceil(log(t, per_branch_max)) if t > 1 else 1
tree_levels = int(n)

if tree_levels == 1:
    x1 = t
```

*Ilustración 16 Algoritmo reajuste de árboles 1*

Se cargan los parámetros del fichero de configuración del proyecto para el mínimo y el máximo número de elementos por rama.

El valor de “count” llega a la función por parámetro y es el número de elementos que componen el índice.

Se redondea por arriba el logaritmo de “t” en base “per_branch_max” si “t” es mayor que 1, si no se le asigna 1 y se convierte a entero para obtener los niveles del árbol.

Si el nivel es 1 se tiene una lista de tamaño “t”, siendo este el número de elementos del índice.

```
# case: tree_level > 1
else:
    x = per_branch_min
    for i in xrange(per_branch_max - 1, per_branch_min - 2, -1):
        if log(t, i) > n:
            x = i + 1
            break

    x1 = int(ceil(t / pow(x, n - 1)))
```

*Ilustración 17 Algoritmo reajuste de árboles 2*

En caso de que el nivel del árbol sea mayor a uno ya no se tiene solo una lista, sino una lista de rangos.

Se tiene un bucle que cuenta hacia atrás desde el número máximo de elementos por rama menos uno hasta el número mínimo de elementos menos uno (ya que en el “xrange” de Python el último elemento indicado para el rango no está incluido) y se comprueba si el logaritmo de “t” en base a ese número es mayor que “n”. En caso de que se cumpla la condición se suma uno al número y con ello se obtiene “x”, que es el tamaño del nodo padre.

A partir de “x” se calcula “x1” siendo “x1” el tamaño que tendrá la lista actual. Para obtener “x1” se convierte a entero el resultado de redondear hacia arriba la división entre el número de elementos del índice y el tamaño de la lista padre elevado a “n”-1.

Se trabaja con enteros ya que las listas siempre deben tener un número entero de elementos.

### 4.3.3 Creación de los Ficheros.

A continuación se mostrará la función de creación de un tipo de fichero. Las funciones de los demás tipos de fichero son muy similares, adaptándose en cada caso a los tipos de objetos necesarios en cada plantilla y otras particularidades.

```
@staticmethod
def _create_profiles_branch(lang_code, letters, profile_list, partial_item):

    if lang_code == LangCode.es_ES.name:
        template = 'seo_letters_es.tmpl'
    elif lang_code == LangCode.pt_BR.name:
        template = 'seo_letters_pt.tmpl'
    else:
        template = 'seo_letters_en.tmpl'

    t = Template(file=os.path.join(os.path.dirname(os.path.abspath(__file__)), '..', '..', '..', 'templates',
                                   'generated', template), searchList=[{'profile_list': profile_list,
                                                                        'letters_lang': letters,
                                                                        'partial_item': partial_item}])

    html_min = htmlmin.minify(unicode(t), remove_comments=True, remove_all_empty_space=True,
                              remove_optional_attribute_quotes=False).encode('utf-8')

    filename = profile_list.uri.replace('/', '-')
    code = sharding_code(profile_list.subdomain + '_' + profile_list.uri)

    path_files = PathFiles()
    path = path_files.path + path_from_code(code) + '/user_' + profile_list.subdomain + '_' + filename + '.html.gz'

    save_gzip_file(path, html_min)
```

*Ilustración 18 Función de creación de ficheros*

A la función le llegan los parámetros “lang_code”, “letters”, “profile_list” y “partial_item”.

El parámetro “lang_code” se utiliza para seleccionar el idioma de la plantilla a utilizar.

El parámetro “letters” se utiliza dentro de la plantilla para saber en qué letras hay contenido y poner los enlaces correspondientes en ellos, ya que el árbol empieza en una letra, ya sea de grupo o de usuario, según el tipo de índice.

El parámetro “partial_item” se utiliza para rellenar la información necesaria en los campos de los partials que incluye la plantilla, como puede ser la cabecera y el pie de página.

El parámetro “profile_list” se trata del objeto principal que obtiene la mayor parte de información utilizada en la plantilla.

La función “Template” se obtiene de la librería de CheetahTemplate y es la que se encarga de rellenar la plantilla, pasándole la plantilla a generar y los objetos con los que se debe rellenar. Aparte de objetos, a la plantilla también se le pueden pasar funciones, como se verá más adelante en la parte de traducción de plantillas.

Una vez obtenido el fichero html a generar se le pasa una función de minimización que se encarga de eliminar espacios innecesarios, comentarios y saltos de líneas para ahorrar espacio.

Llegado a este punto se obtiene la ruta en la que se va a guardar el fichero a partir de la URI del objeto principal, que en este caso se trata de una lista de perfiles de usuario ya que el fichero a generar es la última rama del árbol, y se guarda el fichero pre-comprimido.

### 4.3.4 Esquema de la Base de Datos del Proyecto.

La estructura de la base de datos dentro del proyecto contiene una tabla por tipo de directorio que entre otros datos contiene el shard al que pertenece el elemento en cuestión, y otra tabla que almacena las fechas en las que se ha creado o actualizado cada shard de ese directorio.

La tabla de cada tipo de directorio esta particionado por shard, desde 00 hasta FF en hexadecimal, teniendo 256 shards:

```
CREATE TABLE user_last_date
(
 sharding_date DATETIME,
 sharding CHAR(2),
 PRIMARY KEY (sharding)
);

CREATE TABLE user_info
(
 user_id INT NOT NULL,
 permalink VARCHAR(300),
 name VARCHAR(255),
 letter CHAR(1),
 last_name VARCHAR(255),
 subdomain VARCHAR(3),
 sharding CHAR(2),
 hive INT,
 country INT,
 province INT,
 create_date DATETIME,
 update_date DATETIME,
 quality CHAR(1),
 unique_length INT,
 job VARCHAR(255),
 hash_file VARCHAR(32),
 photo_crop VARCHAR(128),
 vip CHAR(1),
 PRIMARY KEY (user_id, sharding)
)
 PARTITION BY LIST COLUMNS(sharding) (
      PARTITION p00 VALUES IN('00'),
      {...}
      PARTITION pfe VALUES IN('fe'),
      PARTITION pff VALUES IN('ff')
```

*Ilustración 19 Estructura base de datos 1*

De los datos que se observan en la tabla la información referente al usuario, como puede ser el nombre o el país, se extrae de la base de datos principal de la empresa y en el proyecto se añade información necesaria para la creación de las fichas o los sitemaps en dependiendo del caso.

Además se tienen las particiones por sharding code, que nos permiten distribuir los datos, al igual que se ha realizado con las fichas distribuidas por el mismo en el directorio en el que se almacenan.

Además estas tablas cuentan con unos índices que aceleran el acceso a los datos y en algunos casos las operaciones necesarias no se podrían llevar a cabo sin ellos:

```
# Tree LETTER USERS, SITEMAP USER TREE
create index ui_slnl_idx on user_info(subdomain,letter,name,last_name);
create index ui_lnl_idx on user_info(letter,name,last_name);
# Tree SITEMAP USER LEAF
create index ui_sc_idx on user_info(subdomain, create_date);
# Tree HIVE_LETTER USERS SITEMAP USER TREE
create index ui_shlnl_idx on user_info(subdomain,hive,letter,name,last_name);
create index ui_hlnl_idx on user_info(hive,letter,name,last_name);
# Tree HIVE_PROVINCE USERS (not www), SITEMAP USER TREE
create index ui_shpnl_idx on user_info(subdomain,hive,province,name,last_name);
# Tree HIVE_COUNTRY USERS (www), SITEMAP USER TREE
create index ui_hcnl_idx on user_info(hive,country,name,last_name);
```

*Ilustración 20 Estructura base de datos 2*

En este caso se muestran los índices creados sobre la tabla “user_info”, los cuales son necesarios tanto para la creación de distintos tipos de árboles, como para los sitemaps de las hojas del árbol y los sitemaps de los árboles mismos.

Por ejemplo el índice “ui_sc_idx” se crea sobre las columnas “subdomain” y “create_date” y se utiliza para la creación de los sitemaps de las fichas de usuario por subdominio. Para la creación del sitemap se necesita hacer una consulta ordenando los datos por fecha de creación, y dado que en el subdominio “www” se incluyen todos los usuarios y que se cuenta con más de diez millones de usuarios, esta operación no se podría realizar sin contar con el índice ya que la base de datos tendría que hacer la ordenación en el momento y se quedaría sin memoria.

Por otro lado también se cuentan con procedimientos “upsert” para estas tablas, que se encargan de comprobar si el registro existe y lo actualiza o en caso contrario lo crea.

```
drop procedure if exists upsert_content_info;
DELIMITER //
create procedure upsert_content_info
(in    p_content_id            INT,
 in    p_sharding              CHAR(2),
 in    p_subdomain             INT,
 in    p_create_date           DATETIME,
 in    p_update_date           DATETIME)

begin
  if (select count(content_id) from content_info where user_id = p_user_id and sharding = p_sharding) > 0 then
    begin
      /* update user */
      update content_info set subdomain=p_subdomain, create_date=p_create_date, update_date=p_update_date
      where content_id = p_content_id and sharding = p_sharding and update_date < p_update_date;
    end;
  else
    begin
      /*insert user*/
      insert into content_info (content_id, sharding, subdomain, create_date, update_date)
      values (p_content_id, p_sharding, p_subdomain, p_create_date, p_update_date);
    end;
  end if;

end //
DELIMITER ;
```

*Ilustración 21 Estructura base de datos 3*

En este caso se muestra el procedimiento referente a la tabla “content_info” ya que tiene un tamaño más reducido debido a que dispone de menos campos. El resto de procedimientos son parecidos aunque cambia la sentencia “if” ya que los campos y condiciones que se comprueban no son los mismos.

También se cuenta con dos funciones: la primera recibe como parámetro un carácter y si ese carácter es una letra en el rango [a-z] devuelve la misma letra en mayúscula, mientras que si es cualquier otro carácter devuelve el símbolo #, siendo este el que se usa en los índices para los nombres que no empiezan por letra y agrupa todos los demás símbolos; la segunda recibe una cadena de texto y llama a la primera función con la primera letra por la izquierda de dicha cadena, devolviendo el carácter resultante.

### 4.3.5 Generación de los Ficheros de Textos a Traducir y Traducción de las Plantillas.

Para la generación de los archivos .pot con los textos a traducir se ha realizado un script en bash debido a que el programa "xgettext" que genera los ficheros extrayendo el texto a traducir necesita encontrar la función que marca el texto a traducir y esto no era posible con las plantillas en la extensión .tmpl utilizada por CheetahTemplate.

```
#!/bin/bash
if [ $# -ne 1 ];then
    echo "Define path for created .pot file"
fi

DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

for template in `find $DIR/../templates/originals/ -name "*.tmpl"`
do
    cheetah compile $template
done

xgettext --language=Python --keyword=_ --output=$DIR/po_files/peacock.pot \
 --from-code=UTF-8 `find $DIR/../templates/originals/ -name "*.py"`

msgmerge -U $DIR/po_files/peacock_es_ES.po $DIR/po_files/peacock.pot
msgmerge -U $DIR/po_files/peacock_en_US.po $DIR/po_files/peacock.pot
msgmerge -U $DIR/po_files/peacock_pt_BR.po $DIR/po_files/peacock.pot

find $DIR/../templates/originals/ -type f -name '*.py' -delete
```

*Ilustración 22 Script generación ficheros de texto a traducir*

El script recibe por parámetro la ruta en la que se quiere generar el archivo de traducción, busca en el directorio en el que se encuentran las plantillas originales todos los ficheros que acaban en .tmpl y ejecuta el comando "cheetah compile".

Este comando se utiliza para compilar el fichero .tmpl a código Python en el que sí se puede encontrar la función de traducción con lo cual se puede ejecutar el programa "xgettext" sobre ellos.

A continuación se añaden los textos que no estaban de cada idioma al fichero .pot del proyecto y se borran todos los ficheros generados por la compilación de CheetahTemplate.

Dentro de las plantillas originales se necesita el siguiente código para que el programa "xgettext" pueda acceder a la función de traducción y extraer el texto a traducir:

```
#silent _ = $translate($lang_code)
```

Con ello el texto marcado por $_ aplica la función “translate” que es la detectada por el programa “xgettext”.

Para marcar un texto para traducción se realiza de la siguiente manera:

```
<h2 class="encabezado_listado">$_("Services offered")</h2>
```

En este ejemplo se marca el texto “Services offered” como texto para traducir y que aparecerá en el fichero .pot del proyecto de la siguiente manera:

```
#:
/home/odan/Documentos/Proyectos/peacock/i18n/../templates/originals/se
o_user.py:292
msgid "Services offered"
msgstr ""
```

En el fichero aparece a que fichero o ficheros pertenece el texto y que líneas dentro del mismo. Ahora solo faltaría escribir la traducción.

En cuanto a la generación de las plantillas traducidas se tiene un script Python que invoca a una clase generadora con cada tipo de plantilla original disponible y se generan los ficheros .mo a partir de las traducciones de cada idioma:

```
#! /usr/bin/env python
import os
from peacock.enums.lang_code_type import LangCode
from peacock.enums.template_type import TemplateType
from peacock.models.path_translate import PathTranslate
from peacock.translate.templates_generator import TemplatesGenerator


def translate_templates():

    path_translate = PathTranslate()
    path = path_translate.path
    messages = 'LC_MESSAGES'

    for lang in LangCode:
        os.system('msgfmt ' + path + '/po_files/peacock_' + lang.name
+ '.po --output-file ' + path + '/' + lang.name +
                  '/' + messages + '/peacock.mo')

    generator = TemplatesGenerator(TemplateType.user)
    generator.generate('seo_user')
    {…}
```

En cuanto al generador que crea las plantillas traducidas recibe el tipo de plantilla en el constructor, como hemos podido ver en la invocación, y la función “generate” recibe como parámetro el nombre del fichero de la plantilla a traducir.

```
class TemplatesGenerator:
    def __init__(self, template_type):
        self.template_type = template_type

    def generate(self, src_filename):
        file_config = ConfigFile()
        config = file_config.config
        partials_path = config.get('path_partials', 'path')

        if self.template_type == TemplateType.user:
            dest_subfolder = 'user'
        elif self.template_type == TemplateType.job:
            dest_subfolder = 'job'
        elif self.template_type == TemplateType.group:
            dest_subfolder = 'group'
        elif self.template_type == TemplateType.content:
            dest_subfolder = 'content'
        else:
            dest_subfolder = ''

        dest_folder = os.path.join(
            os.path.dirname(os.path.abspath(__file__)),
            '..',
            '..',
            'templates',
            'generated',
            dest_subfolder)

        for lang_code in LangCode:
            lang = lang_code.name.split("_")[0]
            t = Template(
                file=os.path.join(
                        os.path.dirname(os.path.abspath(__file__)),
                        '..',
                        '..',
                        'templates',
                        'originals', src_filename + '.tmpl'),
                searchList=[
                    {'lang_code': lang_code.name},
                    {'lang': lang},
                    {'partials_path': partials_path}]
            )
            t.translate = translate
            with open(dest_folder + '/' + src_filename + '_' + lang +
'.tmpl', 'w') as f:
                f.write(str(t))
            f.close()
```

Se accede al fichero de configuración para obtener la ruta de los partials, y generamos un fichero por idioma con la plantilla traducida, añadiendo el idioma al nombre del fichero.

Un dato a tener en cuenta es que usamos el mismo motor de plantillas para generar las plantillas generadas, con lo cual en las plantillas originales hay que escapar todas las funcionalidades que detecta el motor para ya que si no daría error por falta de objetos

recibidos, debido a que estaría esperando los objetos que se le pasaran a la plantilla ya traducida. Lo único que no hay que escapar es la función de traducción y los "include" ya que si se le pasa la ruta de los partials. Esto quedaría de la siguiente manera:

```
\#if \$user.like_minded_users
    <article class="wr_bloque_blanco wr_box_shadow">
        <div class="bloque_blanco padding10 wr_relacionados_ficha">
            <h3 class="encabezado_listado">$_("People with
affinity")</h3>
            <ol>
            \#for \$like_minded_user in \$user.like_minded_users
               {…}
             \#end for
\#end if
```

Como se observa en la imagen el carácter de escape para CheetahTemplate es "\", y se utiliza "$" para acceder a objetos y "#" para comandos internos como bucles o definición de variables. Todos ellos están escapados salvo "$_" que es la función de traducción.

Aquí se puede ver también cómo se puede trabajar con CheetahTemplates con funciones Python y recorrer listas de objetos, lo que ha sido muy útil y cómodo a la hora de realizar el proyecto.

### 4.3.6 Modos de Ejecución y Fichero main.py

Como se ha comentado anteriormente el programa cuenta con dos modos de ejecución, aunque estos solo están disponibles cuando se están generando fichas (nodos hoja) ya que son los que son propensos a cambiar.

Estos modos no están disponibles para la generación de los índices debido a que por un lado son mucho más rápidos de generar y por otro lado a partir de la actualización de un elemento el índice podría o no cambiar, dependiendo de que parte de la información de la ficha se actualiza, con lo cual los índices se deben generar desde cero en cualquier caso.

Mediante el modo de ejecución "FULL" se obtienen todos los elementos del directorio que se desea generar desde la base de datos principal obteniéndose un puntero del lado del servidor para ir iterando sobre los resultados y generar todas las hojas. Además se escribe en ficheros temporales .sql las operaciones que para modificar datos en la base de datos del proyecto. Estos ficheros se copiaran comprimidos al directorio "inc" y se importarán a la base de datos solo cuando se vaya a realizar un índice o un sitemap para tener la información actualizada desde la última generación que se hizo.

Mediante el modo de ejecución "INC" se obtiene la información de una tabla de la base de datos principal que contienen la información de los elementos que se han modificado, y una vez generadas las fichas se borran los elementos de dicha tablas. Esta tabla se rellena gracias a unos disparadores de la base de datos que se activan cuando se actualiza la información de un usuario, una oferta o cualquier otro elemento de los que se dispone un directorio, e inserta dicho elemento en dicha tabla. Este mecanismo lo tenemos replicado

para tener tolerancia a fallos, con lo cual tenemos dos tablas que se utilizan como colas, y desde el parámetro de configuración “queue_id” se selecciona la tabla que se desea utilizar, como vimos en el apartado del fichero de configuración. En cuanto a los ficheros temporales .sql funciona igual que el modo “FULL”.

A continuación se comentará el funcionamiento del script principal de la aplicación:

Lo primero que se comprueba es si se van a generar nodos hoja o nodos rama (fichas o índices), si es nodo hoja:

```
if not branchs_node:
    tmp_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                            'tmp', QueueFile.filename + '_' + args.mode.lower() +\
                            '_' + args.directory.lower() + '_' + str(os.getpid()))

    logging.debug("Main options selected: " + args.directory + " leaf generation from "
                  + str(args.init) + " to " + str(args.end) + " code (" + args.mode + " mode)")
    for i in range(args.init, args.end + 1):
        if i % nodes_leaf_count == node_leaf_number:
            code = str(hex(i))
            code = code.split('x')[1]
            if len(code) == 1:
                code = '0' + code

            logging.info("Starting parse of " + code + " sharding code")
            x = Master(code, args.directory, args.mode)
            x.generate_leafs()
            logging.info("Finished parsing " + code + " sharding code")

            # Move sql file when finish shard (FULL)
            if args.mode == Mode.FULL.name and os.path.isfile(tmp_path):
                logging.debug("Compressing and moving SQL file from tmp to inc directory (FULL method)")
                inc_files = IncData()
                last_path = inc_files.path_files + '/' + args.mode.lower() + '_' +\
                            remove_underscore(args.directory.lower()) + '_' +\
                            socket.gethostname().split('.')[0] + '_' + time.strftime("%Y%m%d%H%M%S") +\
                            '_' + str(os.getpid()) + '_' + code + '.sql.gz'
                with gzip.GzipFile(tmp_path + ".gz", "w") as gz:
                    with open(tmp_path) as inp_file:
                        shutil.copyfileobj(inp_file, gz)
                move(tmp_path + '.gz', last_path)
                os.remove(tmp_path)
                logging.debug("SQL file processed properly (FULL method)")
```

*Ilustración 23 Funcionamiento de main.py 1*

Se obtiene la ruta de los ficheros temporales y se construye un bucle que recorre los shards indicados en los parámetros del programa. Por cada shard se comprueba si se debe ejecutar en la maquina actual o no. En caso de que se deba generar obtenemos el código y creamos una instancia de la clase “Master” indicándole el sharding code, el directorio que se está generando y el modo de ejecución, información que obtenemos de los parámetros. La clase “Master” es la encargada de seleccionar el generador apropiado a partir de la información proporcionada.

Una vez obtenido el generador adecuado para el directorio se llama a la función “generate_leafs” que es la encargada de hacer todas las operaciones necesarias para generar los ficheros.

A continuación se comprueba si el modo de ejecución es “FULL” y hay ficheros en el directorio temporal, se obtienen los datos del directorio “inc” del fichero de configuración, y se añaden los ficheros por sharding come a un fichero comprimido que se guardará en el directorio “inc”, después se borrará el directorio temporal.

```
    # Move sql file when finish process (INC)
    if args.mode == Mode.INC.name and os.path.isfile(tmp_path):
        logging.debug("Moving SQL file from tmp to inc directory (INC method)")
        # Copy tmp file to path inc files
        inc_files = IncData()
        move(tmp_path, inc_files.path_files + '/' + args.mode.lower() +
             '_' + remove_underscore(args.directory.lower()) + '_' +
             socket.gethostname().split('.')[0] + '_' + time.strftime("%Y%m%d%H%M%S") + '_' +
             str(os.getpid()) + '.sql')

        logging.debug("SQL file processed properly (INC method)")
```

*Ilustración 24 Funcionamiento de main.py 2*

Si el modo de ejecución es “INC” y hay ficheros en el directorio temporal, se mueven los ficheros .sql al directorio “inc”, esta vez sin comprimir y sin agrupar por sharding code ya que los ficheros generados son muchos menos, puesto que el incremental se ejecuta cada poco tiempo.

Si los ficheros que se van a generar son nodo rama lo primero que se comprueba es si el árbol a generar es un de tipo “SITEMAP”, en cuyo caso se procede de la siguiente manera:

```
elif branchs_node and args.tree == TreeType.SITEMAP.name:
    if not args.sitemap:
        logging.info("If SITEMAP tree is selected, sitemap parameter is required")
        sys.exit("If SITEMAP tree is selected, sitemap parameter is required")
    logging.debug("Main options selected: " + args.directory +
                  " sitemap generation for subdomain: " + args.subdomain)
    if args.subdomain == 'ALL':
        sub_info = SubdomainInfo()
        for key, subdomain in sub_info.subdomain_dict.iteritems():
            logging.info("Starting sitemap generation for subdomain " + subdomain)
            x = Master(directory=args.directory, subdomain=subdomain)
            x.generate_sitemaps(args.sitemap, args.sitemaptree)
    else:
        logging.info("Starting sitemap generation for subdomain " + args.subdomain)
        x = Master(directory=args.directory, subdomain=args.subdomain)
        x.generate_sitemaps(args.sitemap, args.sitemaptree)
```

*Ilustración 25 Funcionamiento de main.py 3*

Primero se comprueba que se tienen todos los parámetros necesarios para la ejecución, a continuación se comprueba el valor del parámetro “subdomain” ya que si es “ALL” se construye un bucle lanzando una iteración por subdominio, en caso contrario solo se lanza la generación para el subdominio específico.

En ambos casos se construye una instancia de la clase Master y se lanza la función “generate_sitemaps”, en la cual se comprueba si el tipo de sitemap es hoja o árbol y el

tipo de árbol en caso necesario, encargándose de gestionar la generación invocando los métodos necesarios.

Finalmente si se están generando nodos rama y no son del tipo “SITEMAP”:

```
else:
    logging.debug("Main options selected: " + args.directory +
                  " branchs generation for subdomain: " +
                  args.subdomain + " and tree type: " + args.tree)
    node_branch_number = int(config.get('node_info', 'node_branch_number'))
    nodes_branch_count = int(config.get('node_info', 'nodes_branch_count'))
    if args.subdomain == 'ALL':
        sub_info = SubdomainInfo()
        for key, subdomain in sub_info.subdomain_dict.iteritems():
            logging.info("Starting branchs generation for subdomain " + subdomain)
            x = Master(directory=args.directory, subdomain=subdomain)
            x.generate_branches(args.tree, node_branch_number, nodes_branch_count)
    else:
        logging.info("Starting branchs generation for subdomain " + args.subdomain)
        x = Master(directory=args.directory, subdomain=args.subdomain)
        x.generate_branches(args.tree, node_branch_number, nodes_branch_count)

logging.info("Finished main [peacock] program. Directory: " + args.directory.lower())
```

*Ilustración 26 Funcionamiento de main.py 4*

En este caso el funcionamiento es muy parecido a la generación del sitemap, cambiando la función a la que se llama por “generate_branches” y los parámetros que se le pasan, añadiendo la información del número de máquinas y la máquina actual.

# 5. RESULTADOS

## 5.1 Resultados Técnicos

En cuanto a los resultados técnicos de proyecto, se utiliza la estructura de máquinas descrita en el capítulo de estado del arte, y a continuación se va a entrar más en detalle de las características de estas máquinas.

Como se comentó anteriormente aparte de las máquinas en las que se ejecuta el proyecto y las máquinas de las bases de datos se cuentan con balanceadores de carga.

Dos de dichos balanceadores forman parte de la arquitectura del proyecto para el tráfico interno, los cuales funcionan a nivel de aplicación, en la capa 7, sin SSL.

Se cuenta además con otros cinco balanceadores de carga, dos de ellos son balanceadores a nivel de transporte, en la capa 4, que son el punto de entrada desde internet, y los tres restantes son balanceadores a nivel de aplicación y son los responsables de deshacer el protocolo SSL (SSL Offloading).

En cuanto a las máquinas utilizadas estas cuentan con 1 CPU, 2GB de memoria RAM, y Python 2.7.

Además las máquinas destinadas al directorio ocupan 24 GB de almacenamiento cada una, mientras que las de los demás directorios ocupan 10 GB cada una.

Debido a que tenemos una estructura tolerante a fallos tenemos todo el sistema tolerante a fallos.

Para el directorio de usuarios se utilizan 14 máquinas, para el directorio de ofertas de empleo se utilizan 10 máquinas y el directorio de grupos y el directorio de contenido comparten 10 máquinas entre los dos.

La generación de las fichas de usuario se produce a una velocidad aproximada de 15 fichas por segundo por capa proceso lanzado. Las máquinas orientadas a la generación de usuarios son capaces de ejecutar dos procesos del proyecto en paralelo, con lo cual cada máquina genera 30 fichas de por segundo. Además de esto se cuentan con 7 máquinas que funcionan en paralelo, ya que las 7 restantes son las réplicas destinadas a la tolerancia a fallos, con lo cual obtenemos un total de 210 fichas de usuario por segundo. Si en la red social se dispone de aproximadamente doce millones de usuarios la generación de todas las fichas de usuarios tardaría aproximadamente 16 horas.

La generación de las fichas de empleos se produce a una velocidad aproximada de 10 fichas por segundo y al igual que las máquinas de usuario pueden ejecutar dos procesos en paralelo, con lo cual una máquina puede generar 20 fichas de usuario por segundo. Al igual que ocurre con los usuarios, solo 5 de las máquinas pueden trabajar en paralelo ya que los demás están destinadas a la replicación, con lo cual se genera un total de 100 fichas de empleo por segundo. Si en la red social se dispone de aproximadamente dos

millones de ofertas de empleo la generación de todas las fichas tardaría aproximadamente 6 horas.

Para los directorios restantes no hay una media fija de generación ya que la cantidad de información de una publicación varía mucho debido a que pueden ser solo texto, incluir imágenes o videos o enlaces externos a noticias, con lo cual la velocidad depende de esos factores. Lo mismo pasa con el directorio de grupos, ya que depende de la cantidad de integrantes y de publicaciones que existan en él.

A partir de estos datos se observa que el proyecto es distribuible e infinitamente escalable ya que lo único necesario para que sea más rápido o que mantenga la velocidad actual si aumenta el número de fichas es añadir nuevas máquinas y cambiar los parámetros del fichero de configuración.

### 5.2 Resultados Referentes al SEO

A continuación se procede a realizar una comparativa entre los resultados de SEO de los que se disponía hace aproximadamente seis meses cuando comenzó a desarrollarse este proyecto frente a los resultados obtenidos en la fecha actual.

La velocidad de carga de la aplicación web de la que se disponía antes de la realización de éste proyecto era aproximadamente de 2 segundos, mientras que ahora al servir directamente las fichas generadas pre-comprimidas la velocidad de carga de la web ha bajado a aproximadamente 300 milisegundos, lo que es una gran mejora.

Como se ha mencionado anteriormente la velocidad de carga es un factor que tiene un gran impacto en el SEO y en el usuario según un estudio realizado por LightSpeedNow [15] Google envía un 15% de tráfico adicional a si la web es más rápida, y los buscadores como Bing o Yahoo también envían más tráfico.

Respecto a la indexación [16] de las páginas de la web en Google ha habido una gran mejora:


*Ilustración 27 Evolución de la indexación de es.bebee.com*

Aunque por temas de confidencialidad no es posible presentar las cifras exactas, es posible hablar de porcentajes:

- En diciembre de 2015 la red social contaba con un 7,26% de páginas indexadas.
- En junio de 2016 la red social contaba con un 48,74% de páginas indexadas.
- La variación obtenida desde el lanzamiento del proyecto peacock ha sido de +571%, un valor bastante alto que indica que el proyecto está funcionando mejor de lo que había previamente.

Las páginas indexadas son las que tienen la posibilidad de aparecer en el buscador como resultado de una búsqueda.

Además de la indexación [17] otro factor influyente en el SEO que se ha podido medir ha sido la visibilidad en el buscador. Este factor se refiere a las páginas que han sido resultado de una búsqueda y que además han sido visibles para la persona que ha realizado la búsqueda pero sin necesidad de que llegara a entrar en ella. Por ejemplo los resultados de la segunda página de una búsqueda obtienen visibilidad solo si el usuario que ha realizado la búsqueda ha llegado a la segunda página de los resultados, en cambio si se han visualizado solo los resultados mostrados en la primera página los demás resultados no obtienen visibilidad.

Este parámetro está disponible dividido por directorios:

- El directorio de usuarios, que hace referencia a la dirección es.bebee.com/bees/, ha obtenido un aumento de visibilidad del 63,11%.
- El directorio de ofertas de empleo, que hace referencia a la dirección es.bebee.com/jobs/, ha obtenido un aumento de visibilidad del 254%.
- El directorio de grupos, que hace referencia a la dirección es.bebee.com/hives/, ha obtenido un aumento de visibilidad del 503%.

La visibilidad es importante ya que resultados altos demuestran que las paginas están posicionadas entre los primeros resultados y esto es importante por la probabilidad de conversión de clicks debido a que dicha probabilidad es muchísimo más alta en las primeras posiciones como nos describe el artículo “Probabilidades de click en las SERPs de Google” [18].

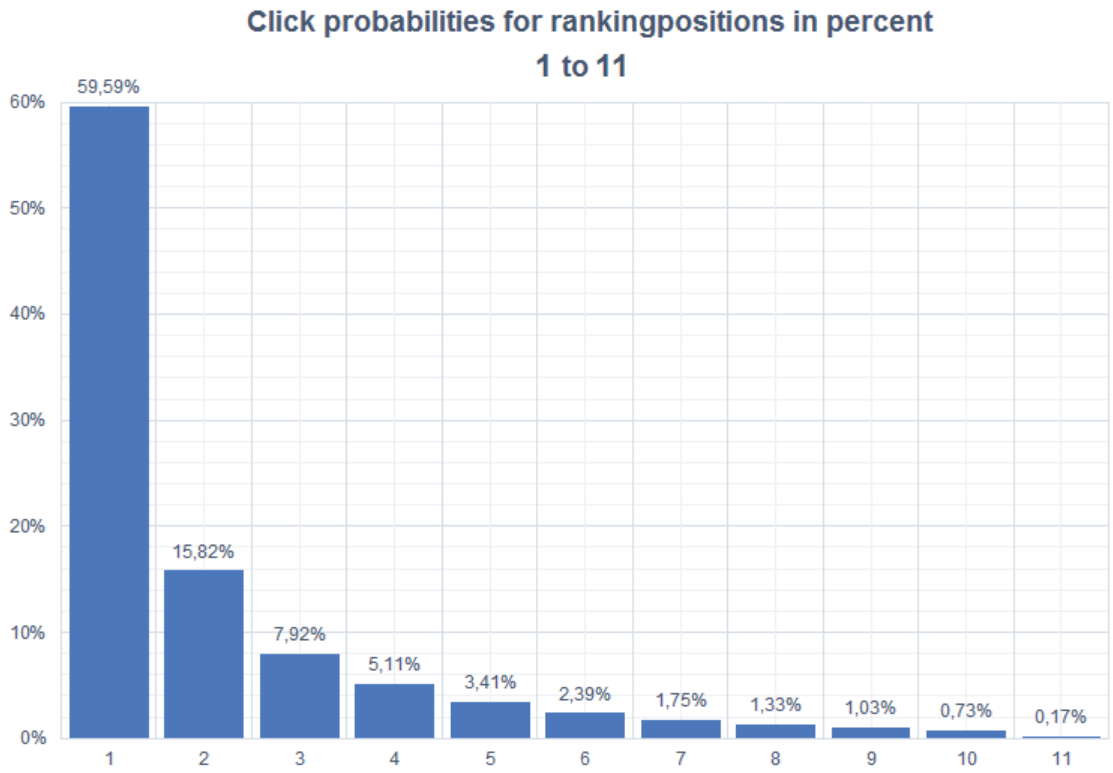


*Ilustración 28 Probabilidades de "click" entre las posiciones 1 y 11 de los resultados de búsqueda*

Otro dato del que se dispone es el tráfico orgánico, que como se mencionó anteriormente es el tráfico no pagado proveniente de las páginas de resultados de un buscador, como se pueden observar en los datos obtenidos de Google Analytics [19]:

- El tráfico orgánico que accede al directorio de usuarios ha aumentado un 58%.
- El tráfico orgánico que accede al directorio de ofertas de empleos ha aumentado un 2583%.
- El tráfico orgánico que accede al directorio de grupos ha aumentado un 866%.
- El tráfico orgánico procedente de Google en general ha aumentado un 64%.


*Ilustración 29 Número de sesiones*

En la ilustración se muestra la evolución del número de sesiones. El número de sesiones indica el número total de veces que los usuarios acceden a la web, en intervalos de 30 minutos. Tras este tiempo, un nuevo acceso será contabilizado como una nueva sesión.

A partir de las sesiones se obtiene el número de conversiones [20]. Una conversión se produce en el momento en el cual el usuario realiza una determinada acción marcada como objetivo, y en este caso el objetivo es que el usuario se registre en la página web.

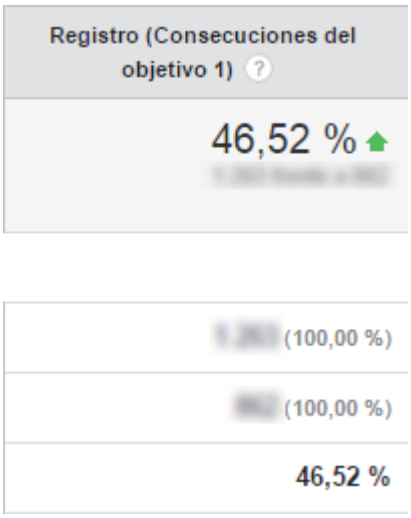


*Ilustración 30 Número de conversiones*

Como se puede observar el número de conversiones ha aumentado un 46,52%.

### 5.3 Resultados Visuales

A continuación se mostrarán a modo de ejemplo algunos resultados de la ejecución del proyecto, siendo estos algunos de los ficheros html generados.

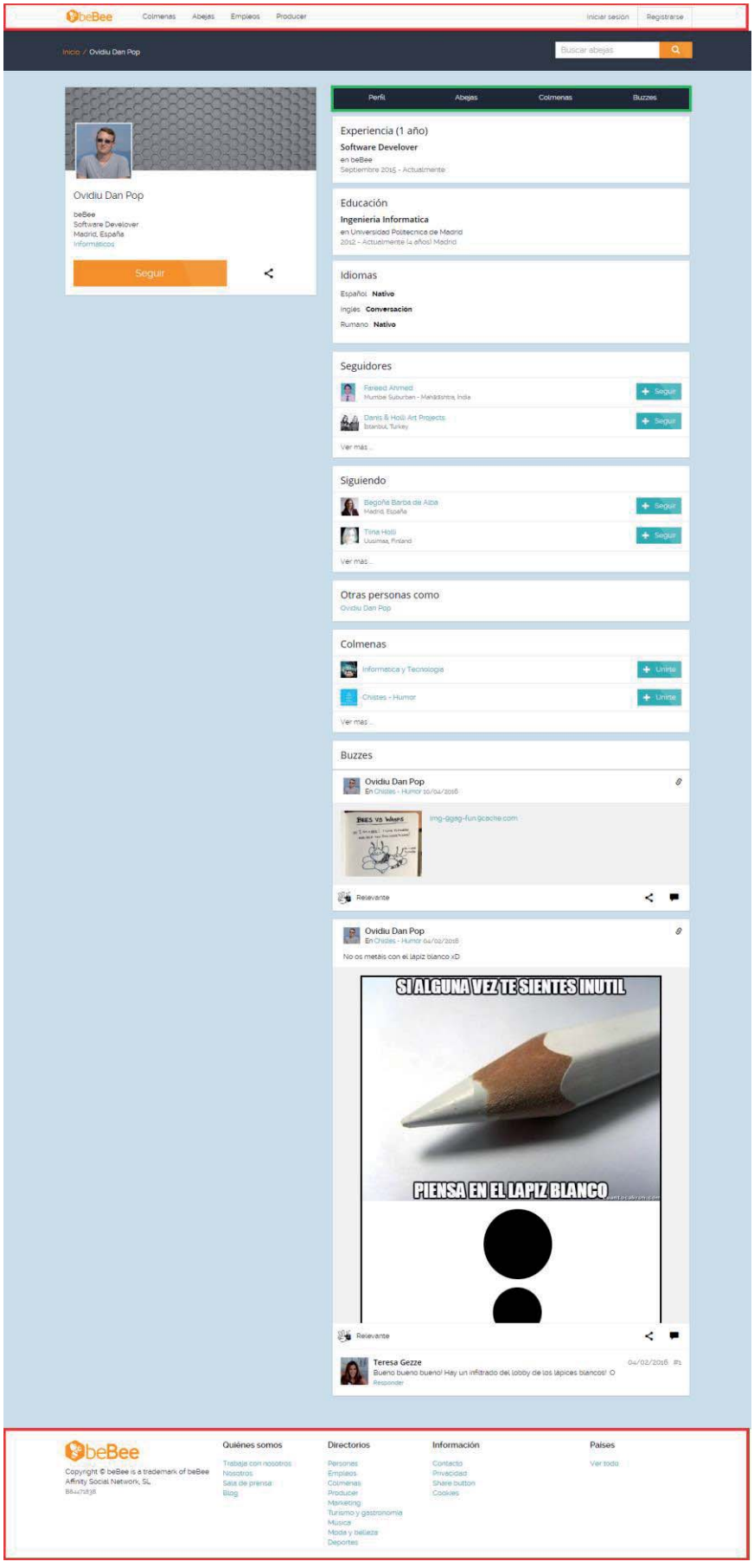
beBee
Colmenas
Abejas
Empleos
Producer
Iniciar sesión
Registrarse
Inicio / Ovidiu Dan Pop
Buscar abejas
Perfil
Abejas
Colmenas
Buzzes
Ovidiu Dan Pop
beBee
Software Developer
Madrid, España
Informáticos
Seguir
Experiencia (1 año)
Software Developer
en beBee
Septiembre 2015 - Actualmente
Educación
Ingenieria Informatica
en Universidad Politecnica de Madrid
2012 - Actualmente (4 años) Madrid
Idiomas
Español Nativo
Inglés Conversación
Rumano Nativo
Seguidores
Fareed Ahmed
Mumbai Suburban - Maharashtra, India
Seguir
Danis & Holli Art Projects
Istanbul, Turkey
Seguir
Ver más...
Siguiendo
Begoña Barba de Alba
Madrid, España
Seguir
Tiina Holli
Uusimaa, Finland
Seguir
Ver mas...
Otras personas como
Ovidiu Dan Pop
Colmenas
Informatica y Tecnologia
Unirse
Chistes - Humor
Unirse
Ver mas...
Buzzes
Ovidiu Dan Pop
En Chistes - Humor 10/04/2016
BEES VS WASPS
img-9gag-fun.9cache.com
Relevante
Ovidiu Dan Pop
En Chistes - Humor 04/02/2016
No os metáis con el lápiz blanco xD
SI ALGUNA VEZ TE SIENTES INUTIL
PIENSA EN EL LAPIZ BLANCO
Relevante
Teresa Gezze
04/02/2016 #1
Bueno bueno bueno! Hay un infiltrado del lobby de los lápices blancos! :O
Responder
beBee
Copyright © beBee is a trademark of beBee Affinity Social Network, SL
B84271838
Quiénes somos
Trabaja con nosotros
Nosotros
Sala de prensa
Blog
Directorios
Personas
Empleos
Colmenas
Producer
Marketing
Turismo y gastronomía
Música
Moda y belleza
Deportes
Información
Contacto
Privacidad
Share button
Cookies
Paises
Ver todo

*Ilustración 31 Perfil de usuario*

En la ilustración se muestra la ficha generada de un perfil de usuario, las secciones marcadas en un rectángulo rojo corresponden con los partials, concretamente en la parte superior se encuentra el partial de la cabecera mientras que en la parte inferior está el pie de página. Estas secciones son las mismas en todas las plantillas, y no se mostrarán en futuras ilustraciones para reducir el tamaño de las ilustraciones.

En la selección verte se encuentran unos botones que desplazan la vista hasta la sección seleccionada de la misma ficha. Los botones como "Seguir", "responder", "relevant", "Unirse" o "ver más" llevan a acción de registro, mientras que los enlaces a colmenas u otros perfiles de usuario dirigen a las fichas de dichos elementos.

También dispone de un buscador realizado en Javascript.

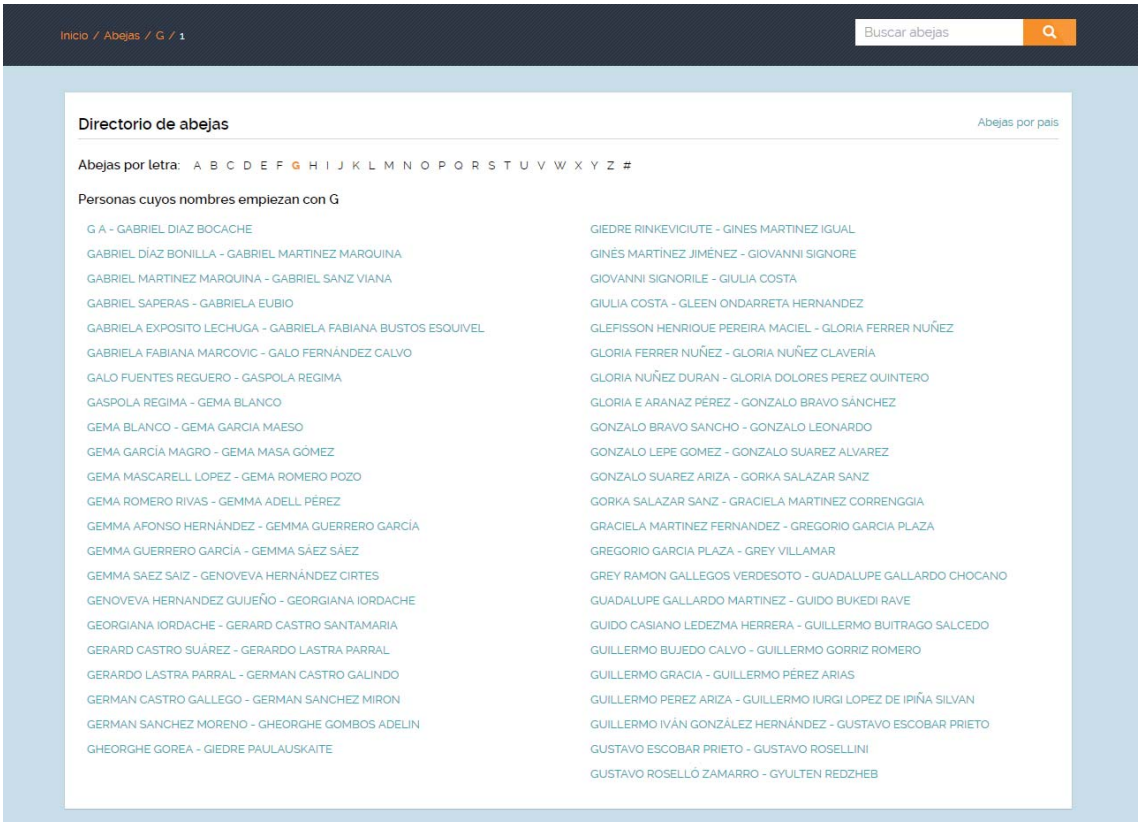


*Ilustración 32 Índice de usuarios por letra*

En la ilustración se muestra el índice de usuarios por letra en el que encontramos rangos de nombres de usuarios con la letra previamente seleccionada. En estas fichas se puede entrar en un rango para acceder al siguiente nivel del árbol, hasta llegar finalmente a una pantalla muy parecida en la que desaparecen los rangos y encontramos una lista con nombres concretos, que redirigen al perfil de la persona en cuestión.

Asimismo se dispone de distintos tipos de índices en los que además de la letra se puede seleccionar la colmena a la que debe pertenecer el usuario o la ubicación del mismo, incluso las tres opciones juntas.

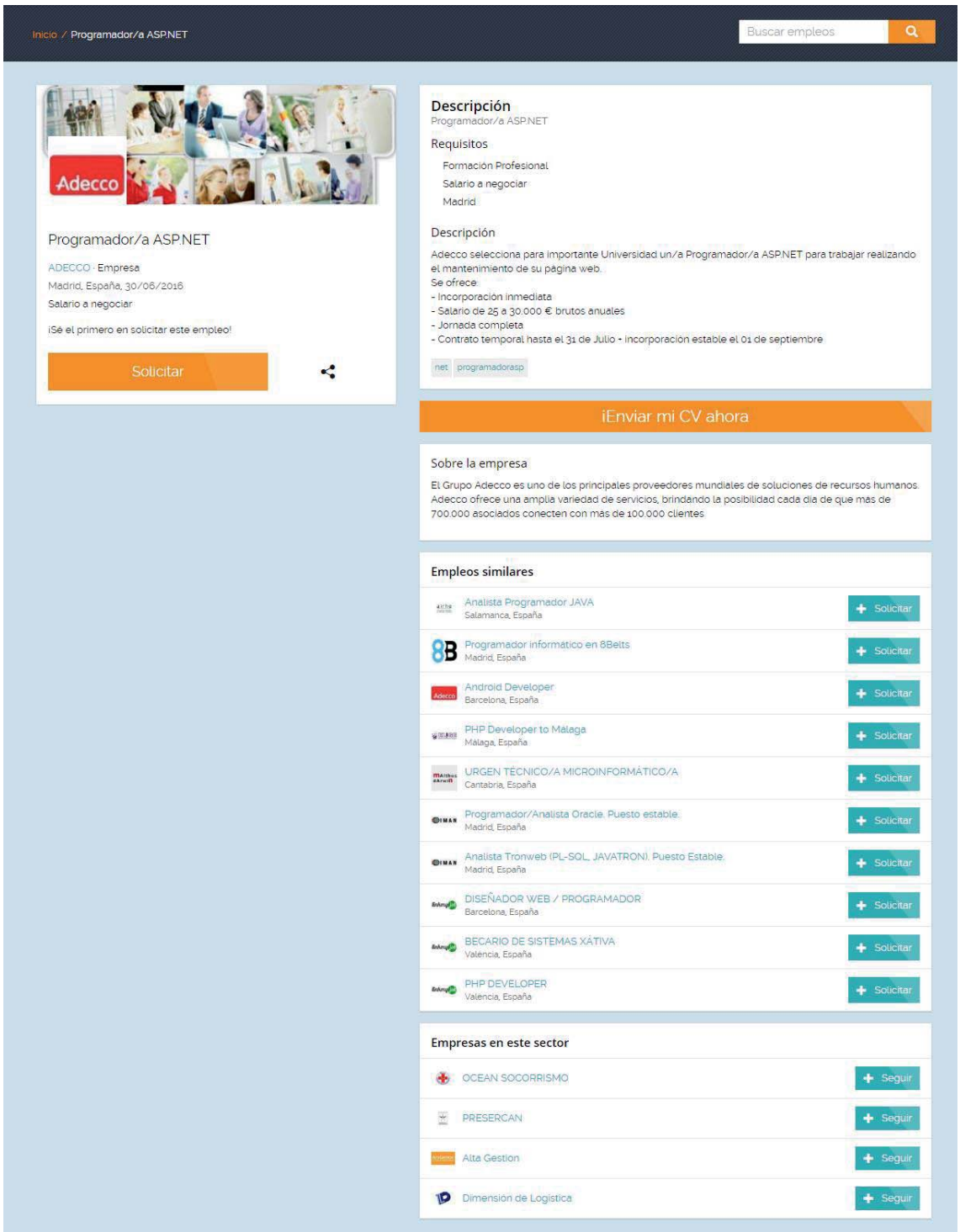


*Ilustración 33 Ficha de oferta de empleo*

En esta ilustración de muestra la ficha de una oferta de empleo. Al igual que en la ficha del usuario los botones “Solicitar”, “Enviar mi CV ahora” y “Seguir” llevan a la acción de registro, mientras que los enlaces a otras ofertas de empleo o a empresas llevan a la ficha de dicho elemento.

La sección de “Empleos similares” se completan con la información de las bases de datos temporales mencionadas anteriormente, que contienen un conjunto de empleos por categoría y se extrae aleatoriamente los empleos que rellenan la sección. Lo mismo ocurre con la sección “Empresas en este sector”.

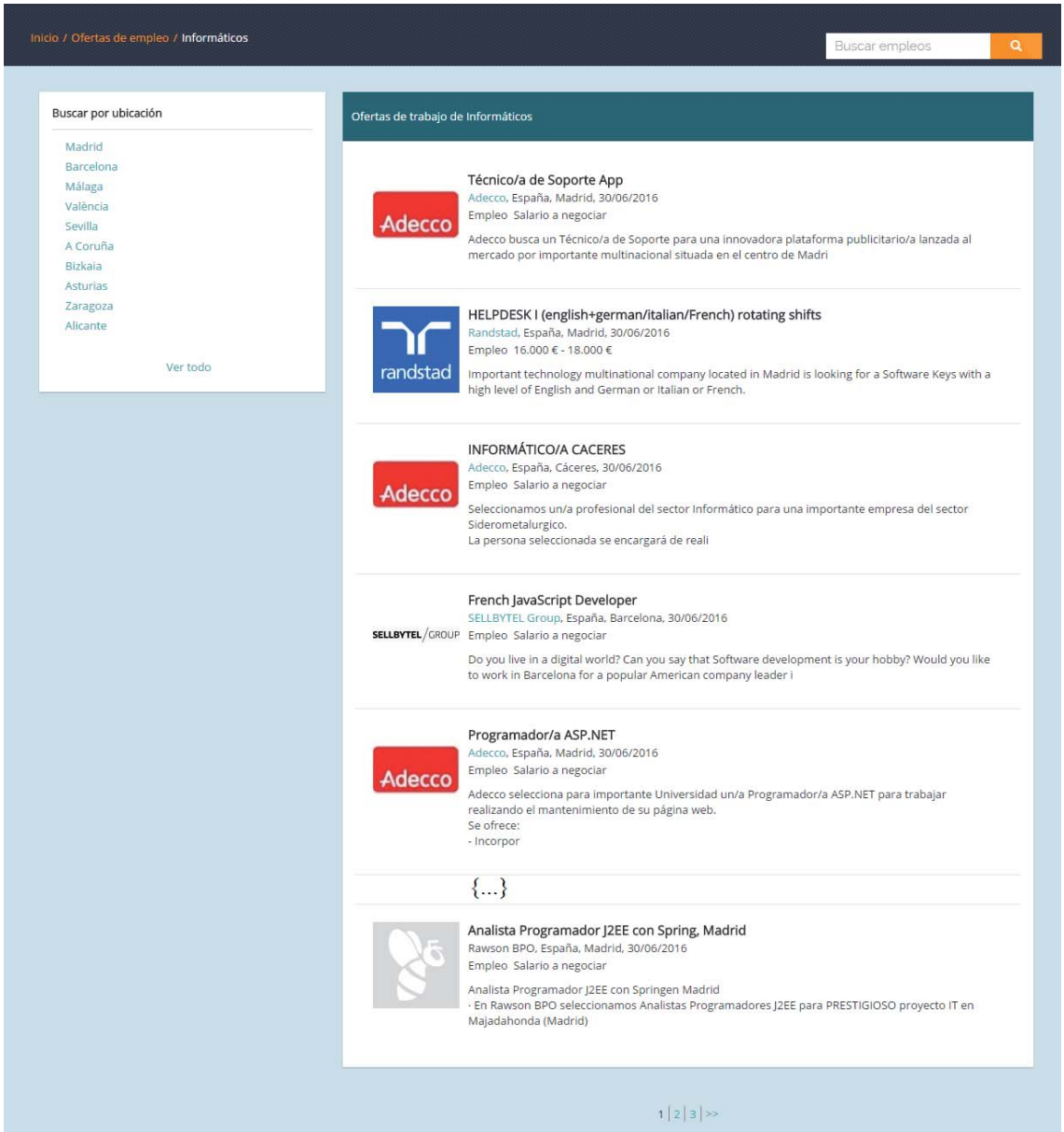


*Ilustración 34 Índice de ofertas de empleo*

En cuanto al índice de ofertas de empleo es como se muestra en la ilustración (aunque esta ha sido recortada debido a que se muestran 50 empleos por página), en este caso no se utiliza una navegación de tipo árbol en la que se navega por rangos, sino que se utiliza paginación. El índice de la ilustración tiene seleccionada previamente una categoría como se puede observar en las “migas de pan”, y también se puede filtrar por ubicación. El botón “Ver todo” lleva a una lista con todas las ubicaciones disponibles, que también está paginada.

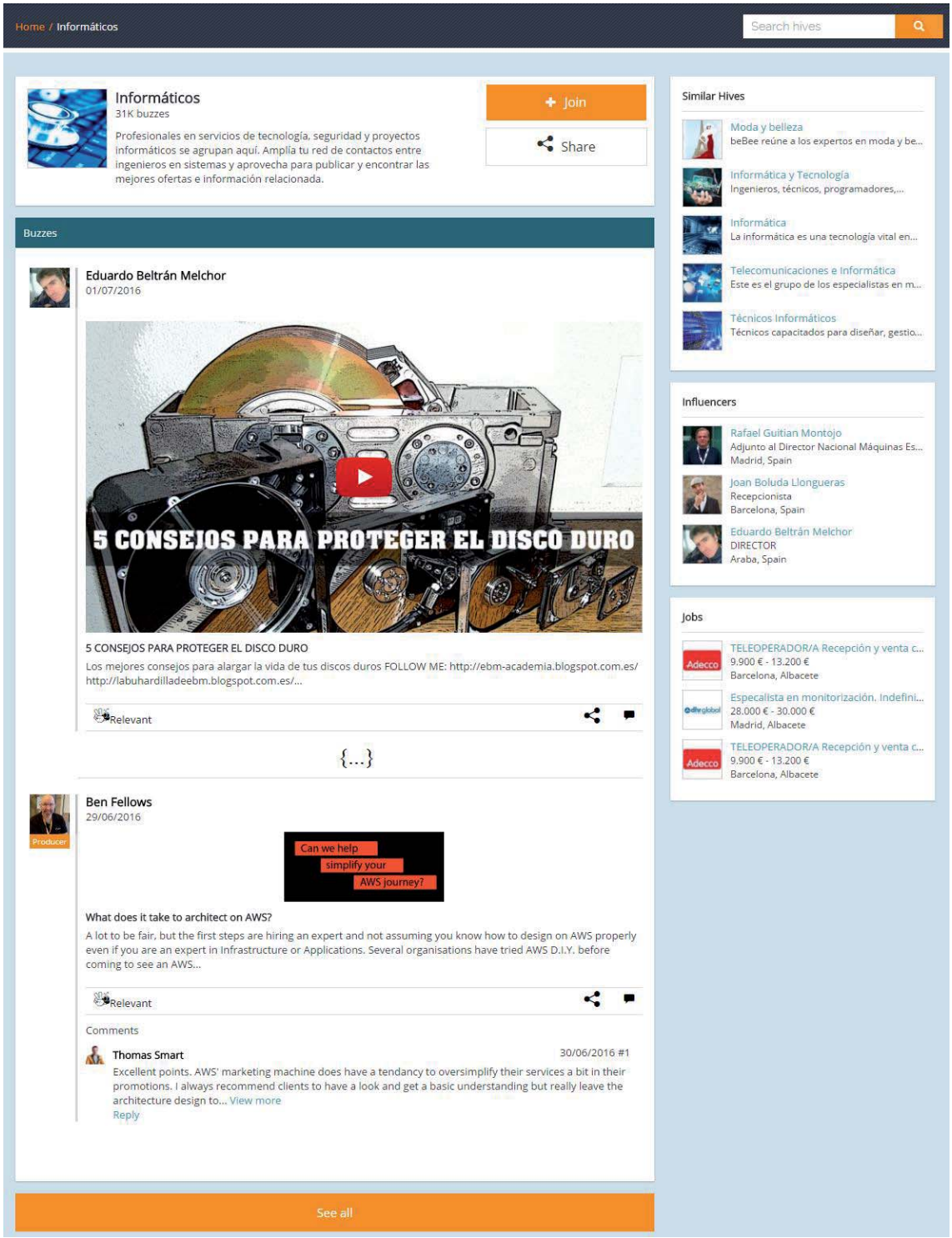


*Ilustración 35 Ficha de un grupo*

En la ilustración se muestra la ficha de un grupo. En este caso cabe destacar que los textos de los comentarios están ocultos si son demasiado largos y que se tiene una función en Javascript para comprimir y expandir el texto. Si se selecciona una publicación se realiza

una redirección a la ficha de dicha publicación, mientras que si se selecciona “See all” se realiza una redirección al índice de contenido por el grupo actual, el cual consta de un listado paginado de todas las publicaciones del grupo.

El índice de los grupos es parecido al índice de usuarios pero se puede filtrar por el idioma del grupo aparte de por la letra con la que comienza el nombre del grupo.

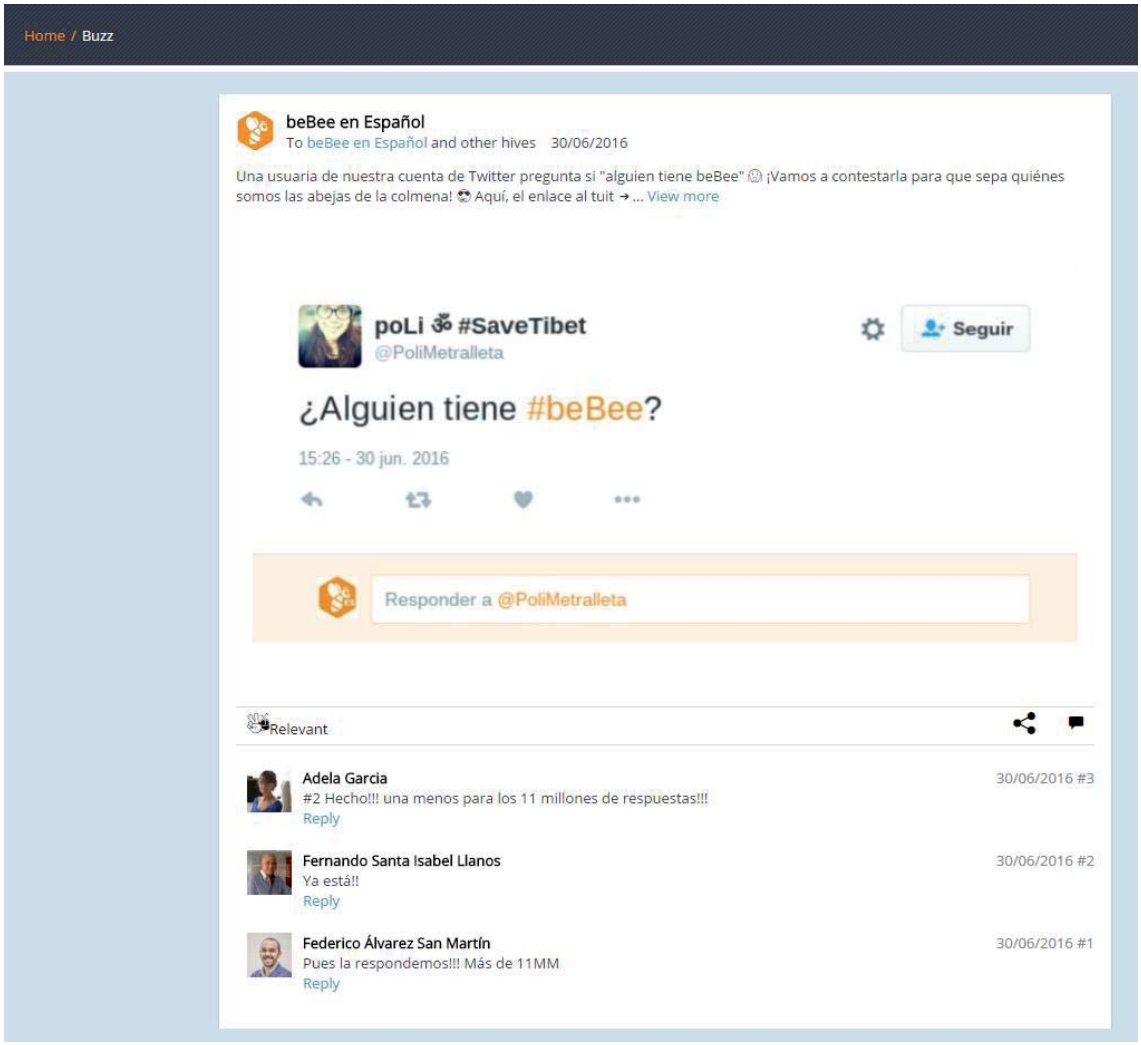


*Ilustración 36 Ficha de una publicación*

En la ilustración se muestra la ficha de una publicación, que es muy parecida a lo que aparece en el listado de publicaciones de la ficha de un grupo, presentando las mismas opciones. El índice de publicaciones, como se ha mencionado anteriormente, se realiza por grupos y consta de un listado paginado de las publicaciones realizadas en un grupo.

## 6. CONCLUSIONES

Se ha logrado que el proyecto cumpla con los objetivos previstos, obteniendo proporcionando un sistema con una velocidad de carga de la página web muy rápida, como se ha indicado en el apartado de resultados, y con ello y las plantillas orientadas al SEO se han mejorado el posicionamiento en buscadores en gran medida.

Además se ha obtenido un sistema distribuible y escalable que proporciona una velocidad de generación de ficheros muy alta. En este apartado también cabe destacar la rapidez con la que trabaja Python, que además dispone de librerías compiladas en C para mejorar su rendimiento que se integran completamente con el lenguaje descrito.

Trabajar con Python ha sido cómodo y cabe destacar que con pocas líneas de código se realizan muchas acciones. Realizar el mismo proyecto en C habría obtenido mayor rendimiento pero hubiera sido más tedioso de implementar y probablemente hubiera llevado mucho más tiempo. Aun así que la aplicación sea capaz de generar doce millones de ficheros en menos de dieciséis horas me parece un buen rendimiento, aunque esto se deba a que se ejecuta en varias máquinas en paralelo.

También se ha observado después de seis meses con al menos una etapa del proyecto en producción que para el SEO el hecho de que sean ficheros estáticos no es un factor positivo, sino todo lo contrario. Este hecho hace que el proyecto tenga una fecha de caducidad, pese a que el proyecto está actualmente en producción y ha cumplido el objetivo de mejorar el SEO en gran medida, para poder mejorar el SEO más de lo que ya se ha conseguido se debe realizar con ficheros dinámicos.

El objetivo de mejorar el SEO es algo complejo debido a que no se conocen todos los factores que influyen en él con seguridad, aunque si se conocen algunos de ellos, como es la velocidad de carga de la página web, en la que se ha centrado este proyecto. Además se conoce que buscadores como Google alteran o modifican su algoritmo de búsqueda con frecuencia, y esto implica cambios en los factores que influyen en el SEO, haciendo que sea una tarea complicada cumplir éste objetivo.

# 7. BIBLIOGRAFÍA


[1] M. Vicedo, «manuelvicedo.com,» [En línea]. Available: http://manuelvicedo.com/marketing-online/generar-trafico-web-cualificado/.

[2] «moz.com,» [En línea]. Available: https://moz.com/learn/seo/internal-link.

[3] «40defiebre,» [En línea]. Available: https://www.40defiebre.com/guia-seo/que-es-seo-por-que-necesito/.

[4] «python.org,» 30 Junio 2016. [En línea]. Available: https://docs.python.org/2/.

[5] J. P. Catalá., «humanlevel.com,» 29 Julio 2013. [En línea]. Available: http://www.humanlevel.com/articulos/desarrollo-web/como-afecta-la-velocidad-de-carga-de-tu-pagina-web-al-seo.html.

[6] Google, «support.google.com,» Google, [En línea]. Available: https://support.google.com/analytics/answer/1009409?hl=es.

[7] B. Klein, «python-course,» [En línea]. Available: http://www.python-course.eu/threads.php.

[8] Cheetah Template, «cheetahtemplate,» [En línea]. Available: http://www.cheetahtemplate.org/.

[9] MySQL, «mysql.com,» [En línea]. Available: http://dev.mysql.com/doc/.

[10] M. Rouse, «searchcloudcomputing.techtarget.com,» Diciembre 2011. [En línea]. Available: http://searchcloudcomputing.techtarget.com/definition/sharding.

[11] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Balanceador_de_carga.

[12] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Transport_Layer_Security.

[13] C. C. R. Andrew J. Hutton, «fedoraproject,» 23-26 Julio 2008. [En línea]. Available: https://ols.fedoraproject.org/OLS/Reprints-2008/kumar-reprint.pdf.

[14] Anonimo, «Wikipedia,» 26 Mayo 2016. [En línea]. Available: https://en.wikipedia.org/wiki/MurmurHash.

[15] C. Munch, «munchweb,» 29 Septiembre 2010. [En línea]. Available: http://munchweb.com/effect-of-website-speed.

[16] Google, «google,» [En línea]. Available: http://www.google.es/intl/es/insidesearch/howsearchworks/crawling-indexing.html.

[17] «jimdo.com,» [En línea]. Available: http://ayuda.jimdo.com/seo/4-herramientas/indexaci%C3%B3n-en-google/.

[18] J. Gonzalez, «SISTRIX,» 11 Febrero 2016. [En línea]. Available: https://www.sistrix.es/blog/probabilidades-de-click-en-las-serps-de-google/.

[19] L. Cloquell, «increnta.com,» 3 Junio 2013. [En línea]. Available: http://increnta.com/es/blog/analitica-web-google-analytics/.

[20] «40defiebre.com,» [En línea]. Available: https://www.40defiebre.com/que-es/ratio-conversion/.

Este documento esta firmado por

| | | |
|---|---|---|
|  | **Firmante** | CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES |
| | **Fecha/Hora** | Mon Jul 04 18:15:47 CEST 2016 |
| | **Emisor del Certificado** | EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES |
| | **Numero de Serie** | 630 |
| | **Metodo** | urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature) |